POLYGONIZATION OF REMOTE SENSING CLASSIFICATION MAPS BY MESH APPROXIMATION

Emmanuel Maggiori[1]*, Yuliya Tarabalka*[1]*, Guillaume Charpiat*[2]*, Pierre Alliez*[1]

[1]Inria Sophia Antipolis-Méditerranée, TITANE team
[2]Inria Saclay, TAO team

ABSTRACT

The ultimate goal of land mapping from remote sensing image classification is to produce polygonal representations of Earth's objects, to be included in geographic information systems. This is most commonly performed by running a pixelwise image classifier and then polygonizing the connected components in the classification map. We here propose a novel polygonization algorithm, which uses a labeled triangular mesh to approximate the input classification maps. The mesh is optimized in terms of an ℓ_1 norm with respect to the classifiers's output. We use a rich set of optimization operators, which includes a vertex relocator, and add a topology preservation strategy. The method outperforms current approaches, yielding better accuracy with fewer vertices.

Index Terms— Remote sensing image classification, polygon generalization, geographic information systems

1. INTRODUCTION

One of the central problems in remote sensing is the assignment of a thematic class to every pixel in a satellite or aerial image, typically referred to as *classification* [1]. One of the most important applications is to integrate the data into geographic information systems (GIS), which requires to represent the detected objects as polygons.

Some object detection techniques in remote sensing imagery directly produce polygonal data, e.g., by fitting rectangles to the image [2]. However, to account for more general shapes one must first classify every pixel and then polygonize the classification map. Moreover, with the advent of deep learning, the pixelwise classification of remote sensing imagery is becoming more and more effective [3, 4]. The usual approach is to vectorize a raster object in a naive way, i.e, by creating a polygon with points at every pixel all around the object boundary, and then to simplify such a polygon. This second step is often referred to as polygon generalization [5].

The most common generalization algorithms can be classified into local and global processing routines. Local routines, such as the radial distance [6] and Reumann-Witkam [7] methods, compare subsequent points in a polygon to decide if one of them may be eliminated. Visvalingam-Whyatt [8] and Douglas-Peucker [9] are global greedy routines that measure the effect of including each point on the entire polygon and not only with respect to the nearest neighbors. These techniques are implemented in most GIS packages (e.g., GRASS, QGIS and ArcGIS), Douglas-Peucker being the most commonly used method by the community [6]. It has been adapted to preserve topology and to generate non-self-intersecting polygons [10, 11].

We here propose a polygonization technique based on the approximation of a triangular mesh to the classification maps. While previous methods simplify based on some measure of distance between the complex and approximated polygon boundaries, our method proceeds in an *integral* manner, i.e., considering the approximation error over the entire surface of the image. In addition, instead of just removing vertices, our approach consists of a rich set of operators which also allows their relocation, permitting to compensate for the imprecisions in the classification map.

2. POLYGONIZATION AS MESH APPOXIMATION

Let us consider a triangular mesh T, consisting of a set of triangles $\{t_i\}$, overlaid on top of an image to partition it (see e.g. Fig. 4). There is a set of class labels $\mathcal{L}$ and we define $C(l, x, y)$ to be the cost associated to assigning a certain label $l \in \mathcal{L}$ to a pixel (x, y) in the image. We also assign a single label l_t to every triangle t. The cost of such an assignment is simply the cost of assigning the label uniformly to all the points inside the triangle. We naturally assume that the label assigned to each triangle is the one with the lowest cost. Our goal is to find the triangulation that minimizes:

$$E(T) = \sum_{t \in T} \left[\min_{l_t \in \mathcal{L}} \iint_{x,y \in t} C(l_t, x, y) dx dy + \lambda \right]. \qquad (1)$$

For each triangle we sum the cost incurred by assigning the optimal label to it, plus an extra weight λ per triangle. Adding λ implies that the mere existence of a triangle has a cost, independently of its label, and is thus used as a regularization term to set the desired coarseness of the mesh.

The authors would like to thank CNES for funding the study. Pléiades images are CNES (2012 and 2013), distribution Airbus DS / SpotImage

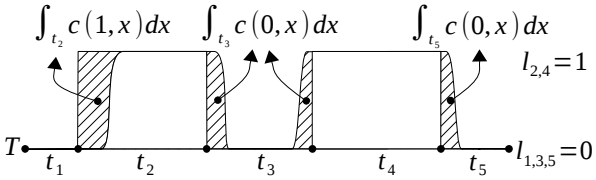


Fig. 1: Illustration of the ℓ_1 cost on a 1-D triangulation.

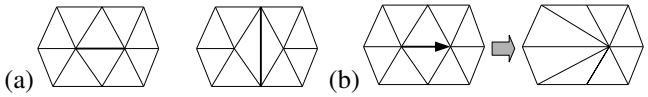


Fig. 2: Operators: edge flip (a) and half-edge collapse (b).

We here consider that a classifier has been used to estimate $P(l,x,y)$, the probability $P(l,x,y)$ of assigning a certain label $l \in \mathcal{L}$ to a point (x,y) in the image (s.t. $\sum_{l\in\mathcal{L}} P(l,x,y) = 1$ and $P(l,x,y) \geq 0, \forall x,y,l$). We here define the cost as follows:

$$C(l,x,y) = ||1 - P(l,x,y)||_1, \tag{2}$$

which can be seen as the volume contained between the classifier's probability surface and the piecewise constant approximation implied by the labeled mesh. Fig. 1 illustrates an example of such cost in 1-D. Let us remark that in addition to using the fuzzy probabilities of a classifier, we can also use a hard classification map (i.e., a unique class assigned to each pixel) by supposing that the probability was set to 1 for the assigned class and 0 for the rest.

The triangular mesh is iteratively optimized by performing a local search, starting from an initial fine lattice. We simulate a number of *changes* that transform the mesh, from T to T', and construct a modifiable priority queue on the energy variation $\Delta E = E(T') - E(T)$ associated to each change. We can restrict the calculation of ΔE to the triangles affected by the change, given the sum over independent triangles in (1). The highest priority change is first extracted from the queue and applied to the mesh. Note that we must relabel the affected triangles and update those elements in the queue that may have been altered as a side effect. This is iterated until there are no changes left. We only consider changes that immediately improve T (i.e., $\Delta E < 0$) and that produce a valid triangulation (e.g., they do not lead to overlapping triangles).

The first two types of changes we consider are the *edge flip* and *half-edge collapse* [12, 13] operators (see Fig. 2). We can transform a triangulation to become any possible simplified triangulation just by flips and collapses, as soon as the vertices are on fixed locations [12], making those two types of operators particularly appealing. However, we also add a *vertex relocation* operator that computes a new position for a certain vertex. This increases the expressiveness of the family of operators, compensating the limitation of flips/collapses that simply recombine predetermined vertices. For example, vertex relocation allows us to start the optimization from a relatively coarse mesh and yet achieve similar (or even better) results than if we departed from a fine mesh (e.g., a vertex on every pixel) and only used flips and collapses. Note that different types of operators can be mixed together in the queue, or they can be applied at will in subsequent stages of the algorithm.

Once the mesh has been optimized, the connected components of triangles with the same class label are outputted as polygonal objects of such class.

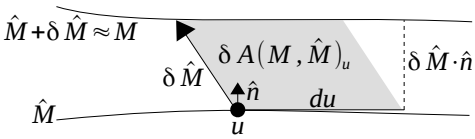


Fig. 3: Elementary area variation δA generated by $\delta\hat{M}$.

3. VERTEX RELOCATION

While the application of flip and collapse moves is self-explanatory, we dedicate this section to our vertex relocation algorithm.

We first see the object boundaries in the classification map as curves in the plane, an object being a connected component of pixels of the same class. We also see the boundaries of objects in the triangulation as curves in the plane. Denoting by M the "real" boundaries on the classification map and by $\hat{M}$ the ones implied by the labeled triangulation, our goal is to modify $\hat{M}$ so that it approaches M. We also use the ℓ_1-norm and seek to minimize the area $A(M,\hat{M})$ contained between the two curves. We adapt the algorithm from [14], where a volume criterion was used to simplify 3-D meshes.

Given a curve $\hat{M}$ parametrized by u, we define $\delta\hat{M}(u)$ as the displacement of a point that belongs to such curve (see Fig. 3), with the goal of approximating M. We assume that $\hat{M}$ can be locally approximated by its tangent, and define du to be a small displacement along the tangent. The variation of area $\delta A(M,\hat{M})_u$ incurred by performing the displacement $\delta\hat{M}(u)$ corresponds to the parallelogram generated by $\delta\hat{M}(u)$ and du. The area of such parallelogram is $|\delta\hat{M}(u)\cdot\hat{n}|\cdot du$, being $\hat{n}(u)$ a vector normal to $\hat{M}$. Since the variation of area $\delta A(M,\hat{M})_u$ may be positive or negative (because $\delta\hat{M}(u)$ may approach or move $\hat{M}$ away from M), we define:

$$\delta A(M,\hat{M})_u = \eta(u)(\delta\hat{M}(u)\cdot\hat{n})du, \tag{3}$$

where η returns -1 if if $\hat{n}$ points toward the area embedded between M and $\hat{M}$ and +1 otherwise, since it would be reducing and increasing the area, respectively (assuming $\hat{n}$ is oriented so that $\delta\hat{M}\cdot\hat{n}$ is positive). The total area variation upon applying all displacements is:

$$\delta A(M,\hat{M}) = \int_u \eta(u)(\delta\hat{M}(u)\cdot\hat{n})du. \tag{4}$$

However, while (4) applies to any curve, in our piecewise linear case when we move a vertex X_i the points along adjacent

edges move accordingly. In fact, the $\delta\hat{M}(u)$ of a point inside a segment $\overline{X_a X_b}$ is a linear combination of δX_a and δX_b:

$$\delta\hat{M}(u) = \lambda_a(u)\delta X_a + \lambda_b(u)\delta X_b, \quad (5)$$

where $\lambda_{a,b}(u)$ are shape functions [15] such that λ_i is 1 on X_i and linearly decreases to zero until reaching the following and previous vertices in the curve. This interpolates X_a and X_b based on their relative distance to $\delta\hat{M}(u)$.

If we differentiate (4) with respect to a particular vertex X_i and use (5), we get:

$$\frac{\partial A(M,\hat{M})}{\partial X_i} = \int_{u*} \eta(u)\lambda_i(u)\hat{n}(u)du. \quad (6)$$

We restrict the domain of integration to the points in edges adjacent to X_i (which we denote by $u*$) because only there $\lambda_i(u)$ is nonzero.

Note that the challenge of evaluating (6) reduces to the computation of $\eta(u)$. We evaluate (6) as follows: first we take a discrete number of points along every edge adjacent to X_i. From each of these points we "shoot a ray" to decide to which side lies the curve we want to approach. For example, in the 2-class case we just shoot rays in opposite directions and see which one hits first the 0.5 probability level, where there is a change of classes and hence an object boundary.

The iterative optimization of X_i is performed by gradient descent. Being k the iteration number, we set:

$$X_i^{(k+1)} = X_i^{(k)} - \alpha^{(k)} \frac{\partial A^{(k)}(M,\hat{M})}{\partial X_i}, \quad (7)$$

where $\alpha^{(k)}$ is an adaptive step that starts at $\alpha^{(0)} = \alpha$ and is multiplied by a factor $\gamma < 1$ whenever an oscillation is detected ($\frac{\partial A^{(k)}}{\partial X_i} \cdot \frac{\partial A^{(0)}}{\partial X_i} < 0$).

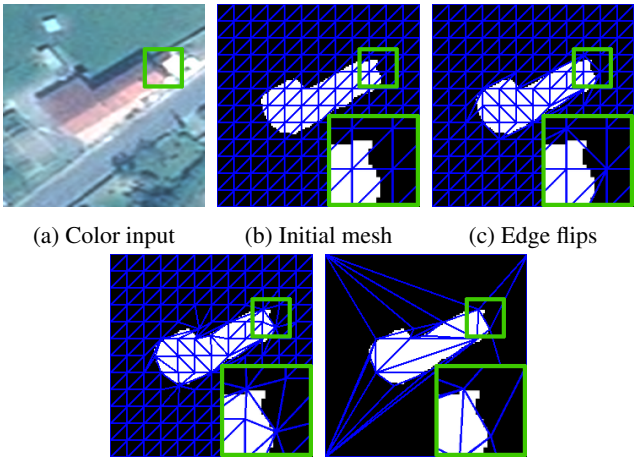
As with the other operators, these moves are simulated and added to the priority queue based on the associated ΔE.

4. TOPOLOGY PRESERVATION

While some mesh approximations may be energetically optimal in terms of (1), they may be unpleasant from a qualitative point of view because they modify the topology of the objects. A notorious case is that nearby buildings are often merged together into one single object, or buildings that were not adjacent in the initial mesh are connected in the simplified mesh.

We deal with this situation by preventing changes that incur in topological changes. To describe the topology of the objects of a certain class we use the Euler characteristic $\chi = V - E + F$, where V, E and F are the number of vertices, edges and faces in a mesh, respectively [16]. For example, when there is a single object of a class, $\chi = 1$, for two separate objects $\chi = 2$, for one object with two holes $\chi = -1$. For each class we must verify that χ does not change as a result of applying an operator. We do this in practice by counting the changes ΔV, ΔE and ΔF incurred by the operator, only doing this locally on the elements that may have changed as a result of its application. We then verify that $\Delta\chi = \Delta V - \Delta E + \Delta F = 0$. We count those vertices, edges and faces that are adjacent to a triangle labeled with the class for which we are verifying the topological change.

Notice that we may adjust the tolerance to topological changes by the coarseness of the initial mesh. For example, if a small hole in the classification map is ignored in the initial labeling, it will remain like that. In other words, one reasonably expects that the initial mesh is finer than what is considered to be the minimum object size.



(a) Color input (b) Initial mesh (c) Edge flips

(d) Vertex relocations (e) Edge collapses

Fig. 4: Effect of applying the different mesh operators.

5. EXPERIMENTS

We experiment on a dataset of Pléiades satellite imagery, manually labeled into two classes: building/not building. We use the classification maps obtained by the two-scale convolutional neural network (CNN) presented in [4].

Let us first illustrate the effect of applying the different operators. Fig. 4(a) shows a piece of color image from the dataset in [4]. In Fig. 4(b) we show the corresponding classification map outputted by the CNN, overlayed with the initial mesh (a lattice with one vertex every 10 pixels). Through the samples we amplify the fragment indicated by the green square on Fig. 4(a). Upon filling the priority queue with flips and applying them till the queue is empty, we get the mesh on Fig. 4(c). On this mesh we perform vertex relocations in a similar fashion, obtaining the mesh in Fig. 4(d). Finally, collapses are done as shown in Fig. 4(e).

We now compare our polygonization method with competing methods. Our algorithm is as follows: we first performs a stage involving only edge flips, followed by a stage of vertex relocations (as in Fig. 4). Then we perform edge collapses and, after each collapse we immediately simulate a

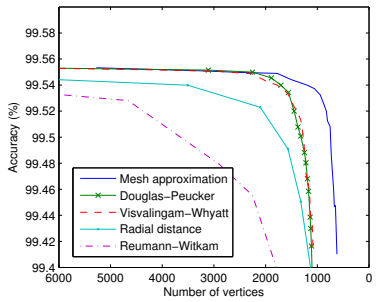
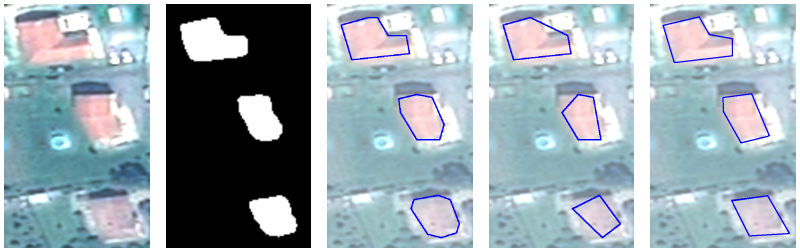


Fig. 5: Performance comparison.



(a) Color input (b) Classif. map (c) DP, thr. 3 (d) DP, thr. 7 (e) Mesh approx.

Fig. 6: Visual comparison ("DP, thr. 3": Douglas Peucker with threshold 3).

vertex relocation on the collapsed node (because we assume that the collapsed node does not end up in the optimal location right away). We can alternatively just mix all the operations together, but we found this way to be more efficient and elegant because the first two stages better align the initial mesh to the classification map at very low cost, prior to start to collapse edges. We start from a fine mesh (one vertex per pixel, in a band around object boundaries).

For vertex relocation we shoot five rays per segment (weighting them based on their distance du) and perform gradient descent with $\alpha = 0.1$, $\gamma = 0.1$, stopping when $\alpha^{(n)} < 0.0001$ or the displacement is below 0.01 pixels.

We compare our polygonization method with the standard techniques used in geographic information systems (GIS). We first vectorize the rasters with GDAL library's function *gdal_polygonize* and then use GRASS and QGIS implementations of the simplification methods mentioned in the introduction.

In Fig. 5 we plot the accuracy of the polygonal approximation as a function of the amount of vertices (e.g., for different values of λ in Eq. 1). This is done on the entire test set of [4]. Since we have ground truth data, accuracy is directly measured as the percentage of correctly classified pixels when rasterizing the polygons. Our algorithm outperforms the other techniques, including the popular Douglas-Peucker and Vasvalingam-Whyatt which are the most accurate among them. The advantage is significant, for example, for a desired accuracy of 99.54%, Douglas-Peucker requires 1200 vertices while our technique achieves the same accuracy with only 700. This is also appreciated visually. Figs. 6(a-b) show a piece of color image and the corresponding classification map. Figs. 6(c) and (d) show the result of applying Douglas-Peucker with two different threshold parameters, and Figs. 6(d) illustrates our results. The parameters of Douglas-Peucker were selected so as to provide a similar accuracy to our method (in Fig. 6c) and a similar amount of vertices than our method (in Fig. 6d). We observe that for a similar accuracy there are too many vertices compared to Fig. 6(d), while for the same number of vertices the polygons by Douglas-Peucker do not represent the underlying objects well, explaining the gap between the curves in Fig 5.

Fig. 7: Examples of color inputs (left) and approximation without (center) and with (right) topological constraints.

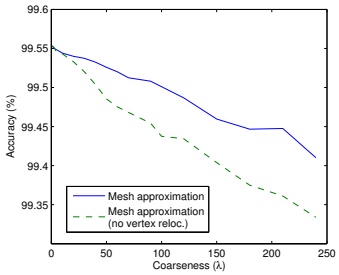


Fig. 8: Approximation with and without vertex relocation.

In Fig. 7 we illustrate the effect of removing the topological constraints described in Sec. 4, where we observe that our methodology effectively outputs polygons that better convey the topology of the underlying objects. Finally, as depicted in Fig. 8, the use of vertex relocation consistently outperforms an equivalent algorithm where vertex relocation is removed.

6. CONCLUDING REMARKS

We presented a mesh approximation algorithm to polygonize remote sensing classification maps. The polygonal objects are more accurate than the current methods used in the GIS community, and provide good approximations of the objects with a significantly lower number of vertices. This is partly because we allow vertices to be located anywhere and not just exactly on the boundary of the original raster objects. Another essential difference of our technique and is that we measure the approximation error in an *integral* manner over the entire surface of the image.

In the future we plan to reinforce regularity relationships (e.g., parallelism, right angles) in the framework, learned from training data, as well as to introduce machine learning in the decision of applying the different operators.

7. REFERENCES

[1] Dengsheng Lu and Qihao Weng, "A survey of image classification methods and techniques for improving classification performance," *International journal of Remote sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[2] Mohammad Awrangjeb, Mehdi Ravanbakhsh, and Clive S Fraser, "Automatic detection of residential buildings using lidar data and multispectral imagery," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 5, pp. 457–467, 2010.

[3] Michele Volpi and Devis Tuia, "Dense semantic labeling of subdecimeter resolution images with convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 881–893, 2017.

[4] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez, "Convolutional neural networks for large-scale remote-sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 645–657, 2017.

[5] Martin Galanda, *Automated polygon generalization in a multi agent system*, Ph.D. thesis, Zurich: Zurich University, 2003.

[6] Wenzhong Shi and ChuiKwan Cheung, "Performance evaluation of line simplification algorithms for vector generalization," *The Cartographic Journal*, vol. 43, no. 1, pp. 27–44, 2006.

[7] K Reumann and APM Witkam, "Optimizing curve segmentation in computer graphics," in *Proceedings of the International Computing Symposium*, 1974, pp. 467–472.

[8] M Visvalingam and JD Whyatt, "Line generalisation by repeated elimination of the smallest area," *The Cartographic Journal*, vol. 30, no. 1, pp. 46–51, 1992.

[9] David H Douglas and Thomas K Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[10] Alan Saalfeld, "Topologically consistent line simplification with the douglas-peucker algorithm," *Cartography and Geographic Information Science*, vol. 26, no. 1, pp. 7–18, 1999.

[11] Shin-Ting Wu and Mercedes Rocío González Márquez, "A non-self-intersection douglas-peucker algorithm," in *SIBGRAPI*. IEEE, 2003, pp. 60–66.

[12] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy, *Polygon mesh processing*, CRC press, 2010.

[13] Antonio Wilson Vieira, Luiz Velho, Hélio Lopes, Geovan Tavares, and Thomas Lewiner, "Fast stellar mesh simplification," in *SIBGRAPI*. IEEE, 2003, pp. 27–34.

[14] Pierre Alliez, Nathalie Laurent, Henri Sanson, and Francis Schmitt, "Mesh approximation using a volume-based metric," in *Proceedings of Seventh Pacific Conference on Computer Graphics and Applications.* IEEE, 1999, pp. 292–301.

[15] Thomas JR Hughes, *The finite element method: linear static and dynamic finite element analysis*, Dover Publications, 2012.

[16] David S Richeson, *Euler's gem: the polyhedron formula and the birth of topology*, Princeton University Press, 2012.